technische universität
dortmund

Fakultät für Informatik
Lehrstuhl 1
Dr. Lars Hildebrand
Iman Kamehkhosh, Marcel Preuß, Henning Timm

Praktikum zu

Einführung in die Informatik für LogWings und WiMas

Wintersemester 2013/14

Übungsblatt 9

Bearbeitungszeit:
13.–17.01.2014

Aufgabe 9.1 – Heaps

Bearbeiten Sie diese Aufgabe auf dem Papier. Beachten Sie die Definition eines Heaps auf der Folie 10 in Kapitel 5, Teil 2, und die Anmerkungen zum Einfügen von Werten in einen Heap ab Folie 19.

a) Fügen Sie nacheinander (mit Hilfe der Einfüge-Operation, die Sie aus der Vorlesung kennen) in einen zu Beginn leeren Heap die Werte 17, 47, 32, 12, 34, 4, 14, 50, 1 und 80 ein.

b) Eine aufsteigend sortierte Folge der Zahlen kann erzeugt werden, indem in jedem Schritt das minimale Element aus dem Heap entfernt wird und dann die Heapeigenschaft wieder hergestellt wird. Führen Sie dies für den gefüllten Heap durch und notieren Sie jedes Vertauschen und jedes Entfernen von Knoten. Sie können aufhören, wenn die 17 entfernt wurde.

Aufgabe 9.2 – Eine einfache Klasse

In einer objekt-orientierten Programmiersprache, können Sie die Informationen (einen Teil der realen Welt), die Sie verarbeiten möchten, durch *Objekte* modellieren. Die Objekte bestehen aus *Attributen* und *Methoden*.

Objekte besitzen Daten, also die Attribute, und Methoden, um diese Daten zu manipulieren oder auszulesen. Durch die Objekte werden die Daten und die Methoden eng miteinander verbunden. Sie können Informationen austauschen (kommunizieren), indem sie gegenseitig Methoden aufrufen. Objekte legen fest, wie durch die Methoden auf die gekapselten Daten zugegriffen werden darf; auf welche dieser Daten andere Objekte zugreifen dürfen.

Eine *Klasse* können Sie sich als Bauanleitung für Objekte vorstellen. In einer Beschreibung einer Klasse, also deren *Quelltext*, legen Sie fest, welche Daten, also Typ und Namen und welche Methoden ein Objekt hat. In dieser Aufgabe schreiben Sie Schritt für Schritt eine einfache Klasse und werden Objekte dieser Klasse *instanziieren* (erzeugen).

a) Die Klasse soll „Kreditkarte“ heißen.

In Java deklarieren Sie jede Klasse in einer eigenen Datei, die genauso wie die Klasse heißt und die Endung `.java` trägt.

```
1  class Kreditkarte {
2    public String inhaberName;
3    public long nummer;
4    public boolean gesperrt;
5
6    public void setzeSperre(boolean gesperrt) {
7     this.gesperrt = gesperrt;
8    }
9
10   public boolean istGesperrt() {
11     return gesperrt;
12   }
13 }
```

Die Zeile 1 wird *Klassenkopf*, die Zeilen 2 bis 12 werden *Klassenrumpf* genannt. Im Klassenrumpf können Variablen (siehe Zeilen 2–5) und Methoden (siehe Zeilen 6 f) deklariert werden.

Bisher haben Sie Variablen nur innerhalb von Methoden deklariert, diese Variablen sind nur innerhalb der jeweiligen Methode sichtbar.

Die Variablen, die innerhalb einer Klasse (außerhalb von Methoden) deklariert werden, heißen *Attribute* eines Objekts oder *Objektvariablen*. Diese Variablen werden für *jedes* einzelne Objekt getrennt angelegt. Sie speichern folglich in jedem Objekt Daten, die völlig unabhängig von den der anderen Objekte sind.

b) Objekte müssen explizit erzeugt werden. Damit der Compiler weiß, wie dies geschehen soll, sollten Sie der Klasse einen *Konstruktor* hinzufügen:

```
6  Kreditkarte(String name, long nummer) {
7    this.inhaberName = name;
8    this.nummer = nummer;
9    this.gesperrt = false;
10 }
```

Beachten Sie, dass der Konstruktor immer genauso heißt wie die Klasse und keinen Rückgabetyp hat, auch kein `void`! In Java beginnen üblicherweise Klassennamen und Konstruktoren mit einem Großbuchstaben und Methoden- und Variablennamen mit einem Kleinbuchstaben. Sie können so Klassen einerseits und Variablen und Methoden einfach auseinanderhalten. Methoden werden häufig mit einem Verb und Variablen mit einem Substantiv bezeichnet.

Lesen Sie auch in den Vorlesungsfolien nach, wozu ein Konstruktor dient und notieren dies kurz in Ihren eigenen Worten:

c) Fügen Sie der Klasse eine Methode `void print()` hinzu, mit der Sie die Werte der Attribute der Klasse ausgeben.

d) Um die Klasse „Kreditkarte “ testen zu können, erzeugen Sie eine weitere Klasse mit dem Namen „TestKreditkarte“.

```
1  class TestKreditkarte{
2    public static void main(String[] args) {
3      Kreditkarte karte;
4      karte=new Kreditkarte("Petra Mustermann",1234567);
5      karte.setzeSperre(true);
6      karte.print();
7      if (karte.gesperrt){
8        System.out.println("Die Karte ist gesperrt.");
9      } else {
10       System.out.println("Die Karte ist nicht gesperrt.");
11     }
12     if (karte.istGesperrt()){
13       System.out.println("Die Karte ist gesperrt.");
14     } else {
15      System.out.println("Die Karte ist nicht gesperrt.");
16     }
17   }
18 }
```

Testen Sie Ihr Programm.

Wie unterscheiden sich die Befehle in den Zeilen 7–11 und 12–16?

e) Löschen Sie im Konstruktor (siehe Aufgabe b)) das Schlüsselwort `this` und den anschließenden Punkt. Das Programm lässt sich kompilieren. Vergleichen Sie die Ausgaben zu denen in Aufgabe d). Was bewirkt das Schlüsselwort `this`? In welchen Blöcken sind die Variablen sichtbar?

Aufgabe 9.3 – Welche ist es denn? – Überdecken von Attributen

Analysieren Sie das folgende Programm, bestehend aus zwei Klassen. Kennzeichnen Sie für jede Variable mit dem Bezeichner `a` jeweils die Zeilen, in denen sie sichtbar ist. Überlegen Sie, welche Ausgabe erzeugt wird. Überprüfen Sie anschließend Ihre Vermutung.

```
1  class Ueberdecken {
2    private int a = 7;
3
4    public void methode1() {
5      System.out.println("Methode 1");
6      System.out.println("a=" + a);
7      int a = 10;
8      System.out.println("a=" + a);
9    }
10
11   public void methode2(int a) {
12     System.out.println("Methode 2");
13     System.out.println("a=" + a);
14     a = a + 2;
15     System.out.println("a=" + a);
16   }
17
18   public int methode3() {
19     System.out.println("Methode 3");
20     System.out.println("a=" + a);
21     a++;
22     if (a > 7) {
23       int a = 3;
24       System.out.println("a=" + a);
25     }
26     System.out.println("a=" + a);
27
28     return a;
29   }
30 }
```

```
1  class TestUeberdecken {
2    public static void main(String[] args) {
3      Ueberdecken u = new Ueberdecken();
4      u.methode1();
5      u.methode2(7);
6      int a = 1 + u.methode3();
7      System.out.println("a=" + a);
8    }
9  }
```

Ergänzende Aufgaben

Aufgabe 9.4 – Eine weitere Klasse

a) Schreiben Sie eine zweite Klasse mit dem Namen „Bank“ und mit den Attributen Name, Bankleitzahl und Ort der Bank und einer Methode `void print()`, die die Daten ausgibt.

b) Erweitern Sie Ihre Klasse „Kreditkarte“ um das Attribut Bank, sodass bei dem Konstruktor ein Objekt des Typs `Bank` übergeben wird und von der Methode `void print()` auch die Bankdaten ausgegeben werden.

c) Testen Sie Ihr Programm, indem Sie in der `main`-Methode der Klasse „TestKreditkarte“ zunächst ein Objekt des Typs `Bank` erzeugen und dies zusätzlich dem Konstruktor `Kreditkarte(...)` als Parameter übergeben.

Aufgabe 9.5 – Noch mal Arrays: Sieb des Eratosthenes

Um alle Primzahlen bis zu einer bestimmten Größe zu finden, gibt es ein Verfahren namens *Sieb des Eratosthenes*, das Sie eventuell schon aus der Schule kennen. Angenommen, alle Primzahlen bis zur Größe n sollen bestimmt werden, dann werden zunächst alle Zahlen von eins bis n aufgeschrieben. Danach werden alle Zahlen weggestrichen, die durch zwei teilbar sind, außer der zwei selbst, also $4, 6, 8$ usw. Nun wird mit der nächsten Zahl fortgefahren, die noch nicht gestrichen ist, in diesem Fall der drei. Es werden also alle Vielfachen dieser Zahl weggestrichen, also $6, 9, 12, \ldots$. Dieses Vorgehen wird solange wiederholt, bis keine weiteren Zahlen mehr gestrichen werden können.

Implementieren Sie eine Methode `siebDesEratosthenes`, die als Parameter eine ganze Zahl n erhält und ein Array von Booleans der Länge $n + 1$ zurückliefert, das für jede Zahl Auskunft gibt, ob es sich um eine Primzahl handelt.

Tipp: Legen Sie in Ihrer Methode als erstes ein Array von Booleans der Länge $n + 1$ an und setzen Sie alle Elemente des Arrays auf `true`.

Aufgabe 9.6 – Pascalsches Dreieck

Beim Ausmultiplizieren von Binomen (Terme der Form $(x + y)^n$) treten Koeffizienten auf, die sich nach einem einfachen Schema berechnen lassen. Zum Beispiel erhält man für $n = 3$ folgendes: $(x + y)^3 = x^3 + 3x^2y^1 + 3xy^2 + y^3$. Die Koeffizienten lauten in diesem Fall $1, 3, 3, 1$; sie heißen *Binomialkoeffizienten*. Schreibt man die Koeffizienten, die sich für ein bestimmtes n ergeben, in eine Zeile und notiert alle Zeilen bis zu einem bestimmten n, so erhält man ein Pascalsches Dreieck. Für $n = 6$ sieht es so aus:

```
n
0                       1
1                   1       1
2               1       2       1
3           1       3       3       1
4       1       4       6       4       1
5   1       5       10      10      5       1
6 1   6   15      20      15      6   1
```

Schreiben Sie nun eine Methode `pascalschesDreieck`, die einen ganzzahligen Parameter n hat und ein Pascalsches Dreieck bis zur Zeile n ausgibt. Auf eine schöne grafische Darstellung kommt es hier nicht an, daher reicht es, wenn die Ausgabe für $n = 5$ ungefähr so aussieht:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

Es gibt mehrere Möglichkeiten, diese Aufgabe zu lösen. Sie können zum Beispiel für jede Zeile des Dreiecks ein neues Array anlegen und die jeweils vorherige Zeile benutzen, um die aktuelle Zeile zu füllen. Sie können auch von Anfang an nur ein Array anlegen, das bereits groß genug ist, um alle Koeffizienten der letzten Zeile aufzunehmen, und dieses Array immer wieder überschreiben.